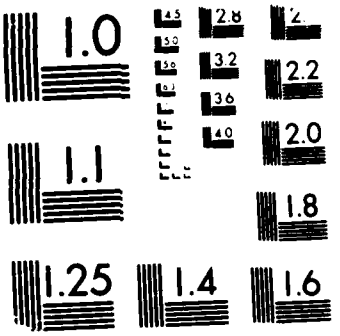
AD-A190 273 CK-LOG SYSTEM CONVERSION (USER'S MANUAL AND USER INTERFACE)(U) JAYCOR VIENNA VA 16 JUN 87 N00014-86-C-2352 1/1

UNCLASSIFIED F/G 12/6 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

AD-A190 273

CK-LOG SYSTEM CONVERSION, USER'S MANUAL & USER INTEFACE

JAYCOR

DTIC
ELECTE
JAN 1 4 1988
S D
D

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

1901 N. Beauregard Street
Suite 503
Alexandria, Virginia 22311-1703

88 1 6 086

CK-LOG SYSTEM CONVERSION, USER'S MANUAL & USER INTEFACE

16 June 1987

DTIC
ELECTE
JAN 14 1988
D

Prepared by:

JAYCOR
1608 Spring Hill Road
Vienna, VA 22180-2270

For:

Naval Center for Applied Research in Artificial Intelligence

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375

In Response to:

Contract No. N00014-86-C-2352
Deliverable A003 & A004

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

1. ELISP to COMMONLISP Conversion

The CK-LOG code written in TOPS-20 ELISP has been converted to Common LISP, and is running on the LMI Lambda. This required both across-the-board syntactic changes in functions as well as entire rewriting of functions. There was no need to change any domain data, as the domain was rebuilt from scratch in Common LISP. The following is an example:

ELISP code

```
(DE LOCALXP (CONTEXT)
      (AND (EQ (NTHCHAR
                  (SETQ CONTEXT
                        ( xseq  (OR CONTEXT CONTEXT!)))
                  1)
            ' 1 )
            (EQ (NTHCHAR CONTEXT 2) ' x )
      )
)
```

COMMON LISP code

```
(DEFUN LOCALXP (CONTEXT)
      (AND (EQ (ELT (STRING (SETQ CONTEXT
                        (x-seq (OR CONTEXT CONTEXT!))))
                  0)
                  ' 1 )
            (EQ (ELT (STRING CONTEXT) 1) ' x )))
```

Because the code is now written in standardized Common LISP, it was transferred to the Symbolics LISP machine with very little difficulty.

2. CK-LOG/Oplan-Consultant User Interface

A user interface/display for Oplan-Consultant was created. It uses both CK-LOG routines and LMI windowing facilities. There are two categories of information that can be displayed: the lattice of time events and the lattice of class information.

Time events are markers in the CK-LOG domain (database of information, or world state) which indicate that some change has occurred at a particular time. The change could be an addition to the domain, a deletion, or a change in a pre-existing condition. CK-LOG is unique in that it associates a time with these changes, thereby allowing for logical reasoning about the feasibility of assertions made to the domain.

The time lattice displays event numbers, representing events in time, in a bottom-up tree. All events fall between **BOTENUM!**, which is represented by **En0**, and **TOPENUM!**, which is represented by **En1000000**. An event can come before or after another event, or just before or just after another event. When an event is before or after another event, events can occur between them. Before and after relationships are represented by a single line connecting two event numbers. When an event is just before or just after another event, no event can occur in between the two events. Just before and just after relationships are represented by a double line connecting two event numbers.

Events can also be incomparable. This could occur when you know that, say, **En1** is after **En0**, and **En2** is after **En0**. This statement contains no information about the relationship between **En1** and **En2**. They would be displayed on the lattice at the same level, both connected to **En0**, but not connected to each other.

Events can occur not only at relative times, but at absolute times as well. The absolute time of **BOTENUM!** is defined at year 0, month 0, day 0, hour 0, minute 0, and second 0. All other absolute times come after this. Events with absolute times can be put before or after each other. They are never incomparable with each other, but can be incomparable with events having no absolute time associated to them.

Events can define intervals, specifying start and stop times. Events can also be added and subtracted from each other. This is useful when increasing or decreasing an absolute time by a certain amount. These operations generate resultant times, which are represented by event numbers in the lattice.

The display of the time lattice can be pruned to show only those events which occur after a given event. In other words, the bottom of the tree need not be **En0**, but can be specified by the user.

The CK-LOG code that creates the time lattice was fully debugged and expanded in the past year. The display of the time lattice was crucial to assisting in the debugging process.

The second category of information that can be displayed is the class lattice. This includes the class hierarchy, dimensions defined on classes, instances of the class, and relations defined on the constants.

A class is a type description for any object entered into the domain. Classes may have subclasses, or specializations. These subclasses inherit all the properties of their parent classes, or generalizations, in addition to having properties of their own. An example of a class is *REGION*, which could have the subclasses *LAND-REGION*, *WATER-REGION*, and *AIR-REGION*. *LAND-REGION* could in turn have the subclasses *PENINSULA*, *COUNTRY*, and *RECTANGLE*. *RECTANGLE* could also be a subclass of *WATER-REGION*, so it would inherit the properties of both *LAND-REGION* and *WATER-REGION*.

These properties are called dimensions. They define what kinds of operations are acceptable for their associated classes, and they set limits on the number of classes that can have this property. For example, **(IS-ADJACENT-TO REGION REGION)** indicates that it's valid to describe one region as being adjacent to another. The maximum number of regions that can have this property is infinite, and the minimum number is 0. The dimension **(STRONGLY-DEFENDS FORCE REGION)** states that a force strongly defends a region. The maximum number of regions a force can strongly defend is infinite, while the minimum number is 0. When defining a dimension, its converse is also created. In this case, the converse dimension would be **(STRONGLY-DEFENDED-BY REGION FORCE)**, stating that a region is strongly defended by a force. The maximum number of forces a region can be strongly defended by is one, while the minimum number is 0.

When an instance of a class, or constant, is created, it represents a real entity in the domain, rather than just a type of entity. The class definition contains limits on the number of instances that can be created. An example of an instance of the class *LAND-*

REGION is *NORTH-PACIFIC*. This is also an instance of the class *WATER-REGIONFR*.

Relations are instantiations of dimensions, with constants plugged in instead of class names. So, assuming that an instance of *REGION* is *ATTU* (this could actually be an instance of *ISLAND*, which is a subclass of *REGION* and thus inherits its dimensions), and assuming that an instance of *FORCE* is *WHITE*, the assertion **(STRONGLY-DEFENDS WHITE ATTU)** could be an acceptable relation in the domain.

The class lattice is represented by a top-down tree. By typing (menu), the user can select the root node of the tree from a pop-up menu listing all of the immediate specializations of the starting root *CLASS*. After selecting the starting class, the user then selects the specific information to be displayed. The choices are the following:

- dimensions: displays the dimensions defined on the chosen class

- inherited dimensions: displays the dimensions defined on the chosen class as well as those inherited from its ancestors

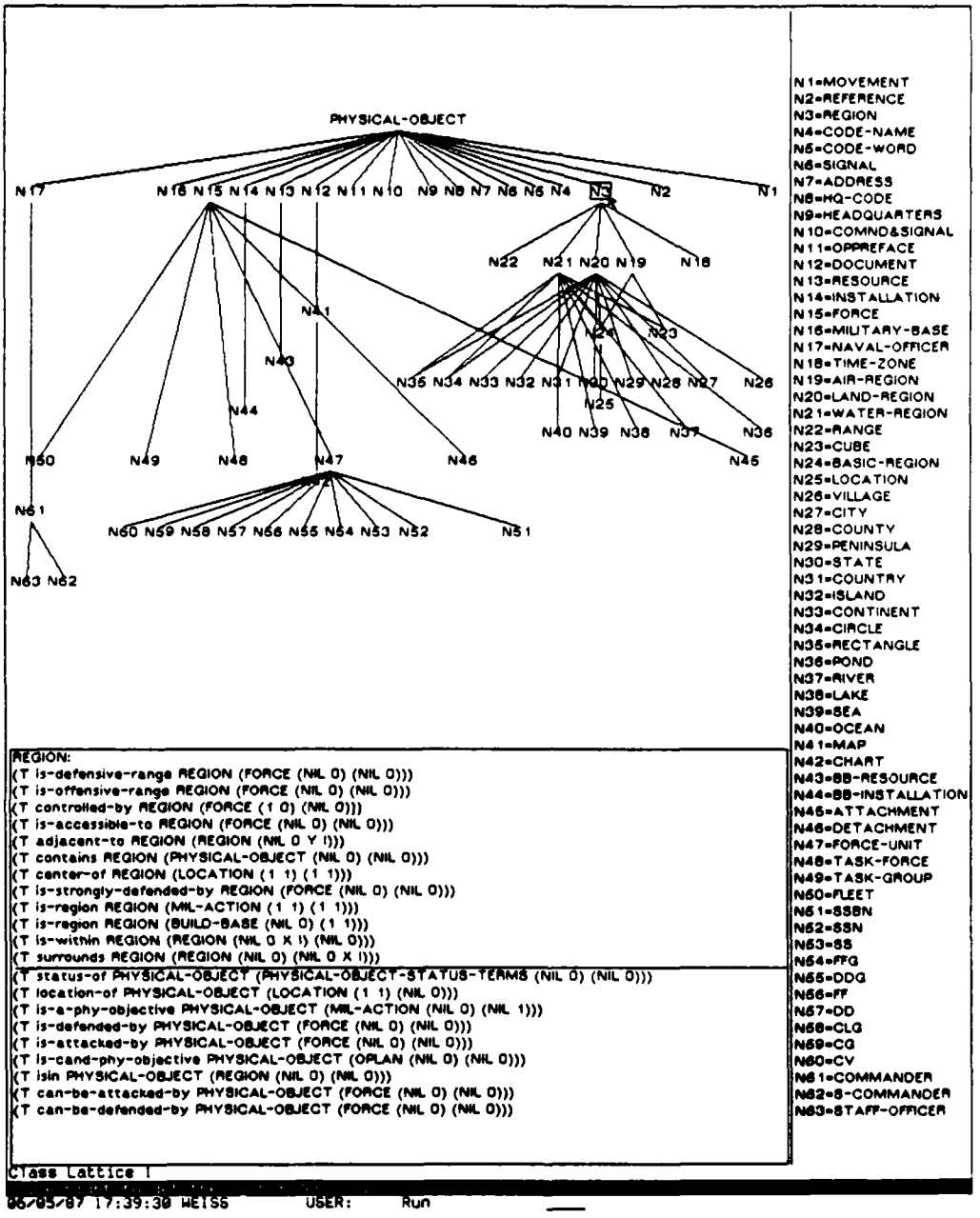
- immediate instances: displays the instances created of the chosen class

- adopted instances: displays the instances created of the chosen class as well as those created of its specializations

Class nodes in the lattice are represented by node numbers, such as **N1**, **N2**, etc. A legend on the right of the screen indicates the class names which correspond to the node numbers in the tree. When the mouse cursor moves over a node number, a box surrounds the node. When the mouse button is pressed over a boxed node, information about that node's class is displayed according to the user's choice of displays. Dimensions are shown at the bottom of the screen. Instances are contained in a pop-up window over the node. The user can then select an instance name by clicking on it, and the relations defined on that instance will appear at the bottom of the screen.

For a more direct approach, the user can type **(display-class <classname>)**. The lattice will show the class hierarchy with **<classname>** as the root, and at the bottom of the screen will be a display of the dimensions and instances defined on the class. By typing (display-constant <constant>), the same information will be displayed, in addition to the relations defined on **<constant>**.

The CK-LOG code for creating classes, dimensions, instances, and relations, was fairly functional when it was translated from the original ELISP. A good amount of debugging was necessary, though, and the display of this information was valuable to the process.



A Typical Display Produced by the User Interface

References

Srinivasan, C.V., *"Advanced Planning Systems & Development of a Planning Consultant for Naval Operational Planning"*, Rutgers University, Department of Computer Science, TR #DCS-TR-197, June 1986.

Srinivasan, C.V., *"Problems, Challenges, and Opportunities in Naval Operational Planning"*, Rutgers University, Department of Computer Science, TR #DCS-TR-187, May 1986.

END

DATE

FILMED

4-88

DTIC